Semiring Programming: A Semantic Framework for Generalized Sum Product Problems

Semiring Programming: A Semantic Framework for Generalized Sum Product Problems

Vaishak Belle[a,b,*], Luc De Raedt[c]

[a] *School of Informatics, University of Edinburgh, Edinburgh, UK.*
[b] *Alan Turing Institute, London, UK.*
[c] *Department of Computer Science, KU Leuven, Belgium.*

Abstract

To solve hard problems, AI relies on a variety of disciplines such as logic, probabilistic reasoning, machine learning and mathematical programming. Although it is widely accepted that solving real-world problems requires an integration amongst these, contemporary representation methodologies offer little support for this.

In an attempt to alleviate this situation, we position and motivate a new declarative programming framework in this paper. We focus on the semantical foundations in service of providing abstractions of well-known problems such as SAT, Bayesian inference, generative models, learning and convex optimization. Programs are understood in terms of first-order logic structures with semiring labels, which allows us to freely combine and integrate problems from different AI disciplines and represent non-standard problems over unbounded domains. Thus, the main thrust of this paper is to view such well-known problems through a unified lens in the hope that appropriate solver strategies (exact, approximate, portfolio or hybrid) may emerge that tackle real-world problems in a principled way.

Keywords: Weighted model counting, Declarative Languages, Semantic abstractions, Semiring frameworks

1. Introduction

AI applications, such as robotics and logistics, rely on a variety of disciplines such as logic, probabilistic reasoning, machine learning and mathematical programming. These applications are often described in a combination of natural and mathematical language, and need to be engineered for the individual application.

Declarative formalisms and methods are ubiquitous in AI as they enable re-use and descriptive clarity. Initial approaches, such as that by Kowalski [64], were rooted in logic, but they have eventually engendered an impressive family of languages. In knowledge representation and constraint programming, for example, languages such ASP [21] and Essence [45] are prominent, which use SAT, SMT and MIP technology [9, 83]. In machine learning and probabilistic reasoning, statistical relational learning systems and probabilistic programming languages such as Markov Logic [80], Church [48] and Problog [26] are increasingly used to codify intricate inference and learning tasks. In mathematical programming and optimization, disciplined programming [50] and AMPL [40] have been developed. Finally, the DARPA project *Probabilistic Programming for Advancing Machine Learning* is motivated in the same declarative spirit.[1] Across disciplines in AI, it has become increasingly clear that taming the model building process, admitting reusable descriptions in expressive languages, and providing general but powerful inference engines is essential.

Be that as it may, it is widely accepted that solving real-world AI problems requires an integration of different disciplines. Consider, for example, that a robot may decide its course of action using a SAT-based planner, learn

*Corresponding author. Vaishak Belle was supported by a Royal Society University Research Fellowship. Luc De Raedt was supported by the European Research Council (ERC) Advanced Grant 694980 "SYNTH: Synthesising Inductive Data Models" and the Research Foundations Flanders.

Email addresses: vaishak@ed.ac.uk (Vaishak Belle), luc.deraedt@cs.kuleuven.be (Luc De Raedt)

[1] http://www.darpa.mil/program/probabilistic-programming-for-advancing-machine-learning

about the world using Kalman filters, and grasp objects using geometric optimization technology. But contemporary declarative frameworks offer little support for such universality: knowledge representation and constraint formalisms mostly focus on model generation for discrete problems whilst leveraging abstractions to hierarchically decompose problems, probabilistic programming languages do not handle linear and arithmetic constraints, and finally, optimization frameworks work with linear algebra and algebraic constraints to specify the problem and thus are quite different from the high-level descriptions used in the other disciplines and do not support probabilistic or logical reasoning.

While adhoc approaches may be possible, the resulting integration may not be transparent or principled. So what is lacking here is a universal modeling framework that allows us to declaratively specify problems involving *logic* and *constraints*, *mathematical programs*, as well as *discrete and continuous probability distributions* in a simple, uniform, modular and transparent manner. Such a framework, together with a solver mechanism, would greatly simplify the development and understanding of AI systems with integrated capabilities, and would tame the model building process. Unlike the case of probabilistic programming, however, it is unlikely that a single, generic inference scheme would apply to all problem instances, and so, we think a combination of solver strategies – exact, approximate, portfolio or hybrid (i.e., approaches that treat different computations as independent but communicating processes) – may be needed. Thus, the main thrust of the paradigm is that: (a) at the semantic level, to view well-known problems through a unified lens, and (b) at the computational level, allow suitable solver strategies for such problems and their combinations. When putting (a) and (b) together, the hope would be that real-world problems can be tackled in a principled way. This paper focuses mainly on (a), and leaves (b) to a discussion section with preliminary observations.

There has been recent progress on this front. A key observation made in [59] is that reasoning about possible worlds is fundamental to many tasks in computer science, including dynamic programming [32], constraint programming [16], database theory [51], probabilistic inference [5], probabilistic logic programming [26], and network analysis [7]. In fact, these tasks essentially invoke a version of the sum product problem, and specifically *weighted model counting* [5], but differ in the exact operations carried out over the possible worlds, which can then be recast in the very same way via semirings. The resulting framework, referred to as *algebraic model counting* (AMC) [59], shows that the computation underlying all these tasks can be defined over a certain class of arithmetic circuits. When the weights over the possible worlds can be factorized at the level of propositions, the tasks can be solved via a single algorithm that obtains local solutions and composes them to yield a global solution. The main limitation of the AMC proposal is that the underlying language is propositional logic, and so the semantics is that of classical logic and computations are essentially defined over discrete spaces. It is, however, possible to accord continuous properties to the underlying propositions [89], which may be sufficient in some cases, but it does not allow for arbitrary arithmetical reasoning. So, the AMC proposal offers an attractive generic inference scheme in the propositional setting. The work in [44] is very similar in spirit. Inspired by arithmetic circuits, sum-product networks [79] have recently emerged as a means for computing products of weighted sums of random variables. When certain conditions are imposed on these networks, which amount to *determinism* and *decomposition* over logical connectives [25], computing the partition function and marginals becomes linear in the size of the network. When that algorithmic machinery is extended to semirings, then as in AMC, we obtain a regime for tractable computations of sum-product problems. As with [89], although certain types of continuous properties can be represented in that framework, the underlying language is not native for continuous modelling and arithmetic reasoning.

In this paper, we propose a new declarative framework called *semiring programming* (SP) that attempts to generalize AMC. Our main thesis is to still formulate computations as the sum product problem, but we will rigorously define a semantics that not only defines AMC in *unbounded* domains, but also *non-standard* (e.g., non-monotonic) ones. To put the proposal in perspective, consider that Eugene Freuder [41] famously quipped: "constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it." The underlying idea was summarized in the slogan:

(constraint) program = model + solver

The vision of SP builds upon this equation in that:

(semiring) program = logical theory + semiring + solver

Thus a model is expressed in a logical system in tandem with a semiring and a weight function. As a framework, SP is set up to allow the modeler to freely choose the logical theory (syntax and semantics) and so everything from non-classical logical consequence to real-arithmetic is fair game. Together with a semiring and these weights, a program

computes the *count*. Our task will be to show that the usual suspects from AI disciplines, such as SAT, CSP, Bayesian inference, and convex optimization can be: (a) expressed as a program, and (b) count is a solution to the problem, that is, the count can be a $\{yes, no\}$ answer in SAT, a probability in Bayesian inference, or a bound in convex optimization. In other words, the count is shown to *semantically abstract* challenging AI tasks. We will also demonstrate a variety of more complex problems, such as matrix factorization, and one on compositionality.

From the perspective of AMC and sum-product networks, the proposal is distinct in a number of ways. The most significant difference is that it is not limited to atomic variables over Boolean connectives, but can allow arithmetic languages (e.g., linear or even non-linear arithmetic) over first-order connectives and quantifiers, to naturally represent unbounded domains. We also handle non-standard semantics as well as non-factorized problems; the latter may not immediately permit a local solution, as can be obtained from the decomposition condition in AMC and sum-product networks. This relaxation is necessary to capture the full class of mathematical programs. (That is, certain classes of optimization problems can be decomposed and factorized [5], but it is unlikely that all instances can.) Another significant difference is the consideration of composition. Note that formal languages already implicitly enable the composition of sentences and operations without needing an upgrade to the semantics. This is precisely the sense in which probabilistic programming languages like Church [48] and ProbLog [26] enable composition. However, composition can be considered more broadly. We motivate an element-wise cross-product composition that encourages the concatenation of entities. Other variants may be desirable or needed, which we will also briefly discuss.

However, with this expressive power, we seem to loose any obvious way to provide a generic yet effective inference scheme for problems expressed in the framework. We think this trade-off is reasonable in the short term as the unified lens afforded by SP might be helpful to having a deeper understanding of computational substrates of problems and devise effective solver strategies.

Informally, from an expressiveness viewpoint, SP is designed to be:

- *universal*, in that it can represent classical problems from disciplines ranging from logic to mathematical programming, and it inherits the strengths of both camps;
- *declarative*, which means that domain knowledge can be expressed in a *program* in a natural, human-readable way, and that it is possible to easily cope with changes in requirements and information in a principled manner;
- *generic*, in that it permits instantiations to a particular language – including real arithmetic (as needed in machine learning), quantification and interpreted symbols (as in satisfiability modulo theory, or SMT), and non-classical consequence (as in answer set programming, or ASP) – since these often correspond closely to the kind of problems they attempt to formalize and solve;
- *solver-independent*, in that inference assumes the role of algorithms, and the formulation of the inference problems is separate from the solution strategies;
- *model-theoretic*, in that the semantics of programs is defined using first-order structures, in service of providing meaning to classical model generation problems ranging from SAT to convex optimization.

To reiterate, the aim is to synthesize problems and techniques across important disciplines in AI, towards a modeling framework that allows one to freely combine and integrate a wide range of specifications. To our knowledge, a number of prior proposals have made promising progress towards these goals, but have fallen short in certain critical features; see the penultimate section for discussions. (We will also contract AMC and sum-product networks with SP against these design principles in that section.)

We emphasize that the thrust of the proposal is to rigorously generalize standard AMC, so as to semantically abstract problems arising from model generation in propositional satisfiability, first-order declarative programming, machine learning, data mining, constraint satisfaction, and optimization. In attempting such a generalization, as hinted above, we will have to give up the existing investigations on the use of propositional arithmetic circuits, including the simple evaluation scheme on local solutions informing global ones [59]. That is not surprising given the infinitary nature of the framework, but we hope the framework will provide the necessary foundations for investigating a solver strategy that works on both finite and infinite domains. Moreover, we illustrate the framework using a programming

language inspired by SMT syntax, but this is meant to be a prototypical starting point for a full fledged programming interface to SP. As can be inferred from above, we would consider the following three desiderata to be essential in the design of such an interface:

1. The logical language should be expressive, supporting a rich ontology of types including lists and graphs.
2. The language should have a semantics defined by a measure on a set of worlds respecting semiring operators.
3. The language should provide abstractions for specifying the solutions of (arbitrary) problems involving *logical reasoning*, *discrete-continuous probability distributions* and *discrete-continuous optimization formulations* in a simple, uniform, modular and transparent manner.

The formulation in the sequel will provide insights on how such a language could be obtained.

Note that this language allows a somewhat literal description of the underlying counting problem, and so we see it as a reasonable starting point. It is only one possible realization of the underlying semantics, and in that regard, it is not necessary that the actual programming interface resemble this syntax: users may feel more comfortable using a conventional imperative programming interface, or a functional one, as in Church [48]. Indeed, [77] propose a probabilistic programming language and refer to the use of SP and other algebraic frameworks for generalizing the language's inference algorithm beyond the probability semiring.

This paper is organised as follows. In Section 2, we briefly review logic, weighted model counting and semirings; in Section 3; we introduce the semiring programming framework and illustrate it on a number of examples; in Section 4, we discuss how different semirings can be combined; in Section 5 different strategies for solvers are introduced; and finally, in Sections 6 and 7 we discuss related work and conclude.

2. Environments

2.1. Logical Setup

The framework is developed in a general way, agnostic about the meaning of sentences. We adopt (and assume familiarity with) terminology from predicate logic [33].

Definition 1. A *theory* $\mathcal{T}$ is a triple $(\mathcal{L}, \mathcal{M}, \rhd)$, where $\mathcal{L}$ is a set of sentences called the *language* of the theory, $\mathcal{M}$ a set called the *models* of the theory, and $\rhd$ a subset of $\mathcal{M} \times \mathcal{L}$ called the *satisfiability relation*.

The set $\mathcal{L}$ is implicitly assumed to be defined over a vocabulary $vocab(\mathcal{L})$ of relation and function symbols, each with an associated arity. Constant symbols are 0-ary function symbols. Every $\mathcal{L}$-model $M \in \mathcal{M}$ is a tuple containing a universe $dom(M)$, and a relation (function) for each relation (function) symbol of $vocab(\mathcal{L})$. For relation (constant) symbol p, the relation (universe element) corresponding to p in a model M is denoted p^M. For $\phi \in \mathcal{L}$ and $M \in \mathcal{M}$, we write $M \rhd \phi$ to say that M *satisfies* ϕ. We say a formula ϕ is *valid* iff ϕ is satisfied at every model, which is then written as $\rhd\phi$. We let $\mathcal{M}(\phi)$ denote $\{M \in \mathcal{M} \mid M \rhd \phi\}$. Finally, the set $lits(\mathcal{L})$ denotes the literals in $\mathcal{L}$, and we write $l \in M$ to refer to the $\mathcal{L}$-literals that are satisfied at M.

This formulation is henceforth used to instantiate a particular logical system, such as fragments/extensions of first-order logic, as well as non-classical logical consequence.

Example 2. Suppose $\mathcal{T} = (\mathcal{L}, \mathcal{M}, \rhd)$ is as follows: $\mathcal{L}$ is obtained using a set of 0-ary predicates $\mathcal{P}$ and Boolean connectives, $\mathcal{M}$ are $\{0, 1\}$ assignments to the elements of $\mathcal{P}$, and $\rhd$ denotes satisfaction in propositional logic via the usual inductive rules. Then, we obtain a *propositional theory*.

Example 3. Define a theory $(\mathcal{L}, \mathcal{M}, \rhd)$ where $\mathcal{L}$ is the positive Horn fragment from a finite vocabulary, $\mathcal{M}$ the set of propositional interpretations, and for $M \in \mathcal{M}, \phi \in \mathcal{L}$, define $M \rhd \phi$ iff $M \rhd_{\text{PL}} \phi$ (*i.e.,* satisfaction in propositional logic) and for all $M' \subsetneq M$, it is not the case that $M' \rhd_{\text{PL}} \phi$. This theory can be used to reason about the *minimal models* of a formula.

Example 4. Define a theory $(\mathcal{L}, \mathcal{M}, \rhd)$ where $\mathcal{L}$ is a first-order language involving 0-ary functions $\{c, \ldots, d\}$, inequalities $\leq, \geq, <, >, =, \neq$. Let $\mathcal{M}$ be the set of mappings from $\{c, \ldots, d\}$ to $\mathbb{R}$. We define $M \rhd \phi$ for $M \in \mathcal{M}$ and $\phi \in \mathcal{L}$ as in first-order logic, assuming in particular that $=, <, >, 0, 1, +, \times, /, -$, exponentiation and logarithms have their usual interpretations [9]. (That is,"$1 + 0 = 1$" is true in all models, as is "$x > y \equiv \neg(y > x)$," and so on [12].) This theory can be used to reason about linear arithmetic (i.e., allowing formulas such as $c + d \leq 5$ and $d \leq e$, where c, d, e are integers or reals) and non-linear arithmetic (i.e., allowing formulas such as $c + d^2 \geq e$).

	factorized	non-factorized	non-classical
finite	WMC	logical-integer programming	ASP
infinite	polyhedron volume	convex optimization	first-order ASP

Figure 1: scope of programs

2.2. Weighted Model Counting

Semiring programming draws from the conceptual simplicity of weighted model counting (WMC), which we briefly recap here. WMC is an extension of #SAT, where one simply counts the number of models of a propositional formula [46]. In WMC, one accords a weight to a model in terms of the literals true at the model, and computes the sum of the weights of all models.

Definition 5. Suppose ϕ is a formula from a propositional language $\mathcal{L}$ with a finite vocabulary, and suppose $\mathcal{M}$ is the set of $\mathcal{L}$-models. Suppose $w\colon \mathit{lits}(\mathcal{L}) \to \mathbb{R}^{\geq 0}$ is a weight function. Then:

$$\mathrm{WMC}(\phi, w) = \sum_{M \rhd \phi} \prod_{l \in M} w(l)$$

is called the *weighted model count* (WMC) of ϕ.

Here, in the context of $\mathcal{L}$-models $\mathcal{M}$, we simply write $t = \sum_{M \rhd \phi} u$ to mean $t = \sum_{\{M \in \mathcal{M} | M \rhd \phi\}} u$.

The formulation elegantly decouples the logical sentence from the weight function. In this sense, it is clearly agnostic about how weights are specified in the modeling language, and thus, has emerged as an assembly language for Bayesian networks [23], and probabilistic programs [38], among others.

2.3. Commutative Semirings

Our programming model is based on algebraic structures called *semirings* [65]; the essentials are as follows:

Definition 6. A (commutative) semiring $\mathcal{S}$ is a structure $(\mathbb{S}, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ where $\mathbb{S}$ is a set called the *elements* of the semiring, $\oplus$ and $\otimes$ are associative and commutative, $\mathbf{0}$ is the identity for $\oplus$, and $\mathbf{1}$ is the identity for $\otimes$.

Abusing notation, when the multiplication operator is not used, we simply refer to the triple $(\mathbb{S}, \oplus, \mathbf{0})$ as a semiring.

Example 7. The structure $(\mathbb{N}, +, \times, 0, 1)$ is a commutative semiring in that for every $a, b \in \mathbb{N}$, $a + 0 = a$, $a \times 1 = a$, $a + b = b + a$, and so on.

3. Semiring Programming

In essence, the semiring programming scheme is as follows:

- **Input**: a theory $\mathcal{T} = (\mathcal{L}, \mathcal{M}, \rhd)$, a sentence $\phi \in \mathcal{L}$, a commutative semiring $\mathcal{S}$, and a weight function w.
- **Output**: the *count*, denoted $\#(\phi, w)$.

The scope of these programs is broad, and so we will need different kinds of generality. Roughly, the distinction boils down to: (a) whether the set of models for a formula is *finite* or *infinite*; (b) whether the weight function can be *factorized* over the literals or not (in which case the weight function directly labels the models of a theory); and (c) whether $\rhd$ is defined in a classical (monotonic) manner or not. The thrust of this section is: (i) to show how these distinctions subsume important model generation notions in the literature, and (ii) providing rigorous definitions for the count operator. In terms of organization, we begin with the finite case, before turning to the infinite ones. We present an early preview of some of the models considered in Figure 1.

```
(set-logic PL)
(set-algebra [NAT,max,*,0,1])
(declare-predicate p ())
(declare-predicate q ())
F = (p or q)
(declare-weight (p 1))
(declare-weight ((neg p) 2))
(declare-weight (q 3))
(declare-weight ((neg q) 4))
(count F)
```

Figure 2: most probable explanation

3.1. Finite

Here, we generalize the WMC formulation to semiring labels, but also go beyond classical propositional logic.

Definition 8. We say the theory $\mathcal{T} = (\mathcal{L}, \mathcal{M}, \rhd)$ is *finite* if for every $\phi \in \mathcal{L}$, $\{M \in \mathcal{M} \mid M \rhd \phi\}$ is finite.

So, a propositional language with a finite vocabulary is a finite theory, regardless of (say) standard or minimal models. Similarly, a first-order language with a finite Herbrand base is also a finite theory.

Definition 9. Suppose $\mathcal{T} = (\mathcal{L}, \mathcal{M}, \rhd)$ is a finite theory. Suppose $\mathcal{S} = (\mathbb{S}, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is a commutative semiring. Suppose $w : \mathcal{M} \to \mathbb{S}$. For any $\phi \in \mathcal{L}$, we define:

$$\#(\phi, w) = \bigoplus_{M \rhd \phi} w(M)$$

If $w : \mathit{lits}(\mathcal{L}) \to \mathbb{S}$, then the problem is *factorized*, where:

$$w(M) = \bigotimes_{l \in M} w(l).$$

Essentially, as in WMC, we sum over models and take products of the weights on literals but w.r.t. a particular semiring. Needless to say, we immediately subsume the framework of algebraic model counting (AMC) [59]:

Proposition 10. *Suppose $\mathcal{T} = (\mathcal{L}, \mathcal{M}, \rhd)$ is a finite propositional theory, $\mathcal{S}$ a commutative semiring, and w a factorized weight function. Suppose $\rhd$ is the standard satisfaction relation in propositional logic. Then SP is equivalent to AMC, that is, the computation of every SP instance can be defined as an AMC task and vice versa.*

Let us consider a few examples.

Example 11. We demonstrate SAT and #SAT. Consider a propositional theory $\mathcal{T} = (\mathcal{L}, \mathcal{M}, \rhd)$ where $\mathit{vocab}(\mathcal{L}) = \{p, q\}$, and suppose $\phi = (p \vee q)$. Letting $\mathcal{M}$ be standard $\mathcal{L}$-models, clearly $|\mathcal{M}| = 4$ and $|\{M \in \mathcal{M} \mid M \rhd \phi\}| = 3$. Suppose for every $M \in \mathcal{M}$, w is function such that $w(M) = 1$. For the semiring $(\{0, 1\}, \vee, 0)$:

$$\#(\phi, w) = \bigvee_{M \rhd \phi} w(M) = 1 \vee 1 \vee 1 = 1.$$

Consider the semiring $(\mathbb{N}, +, 0)$ instead. Then

$$\#(\phi, w) = \sum_{M \rhd \phi} w(M) = 1 + 1 + 1 = 3.$$

```
(set-logic QF_LIA;PL)
(set-algebra [NAT,max,0])
(set-type INT={1,...,10})
(declare-function x1 () INT)
(declare-function x2 () INT)
(declare-predicate p1 ())
(declare-predicate p2 ())
F = ((p1 or p2) => 3*x1 <= 4)
G = (p2 => (2*x2 <= 5))
H = ((F and G) and p2)
(declare-weight TRUE x1*x2)
(count H)
```

Figure 3: logical-integer programming

As with Definition 5, the framework is agnostic about the modeling language. But for presentation purposes, programs are sometimes described using a notation inspired by the SMT-LIB standard [8].

Example 12. We demonstrate MPE (most probable explanation) and WMC. Consider the theory, semiring and weight function w from Figure 2, which specifies, for example, a vocabulary of two propositions p and q, $w(p) = 1$ and $w(\neg p) = 2$. In accordance with that semiring, the weight of a model, say $\{p, \neg q\}$, of the formula $\mathtt{F}$ is $1 \times 4 = 4$. Thus, for the semiring $(\mathbb{N}, \max, \times, 0, 1)$, we have:

$$\#(\mathtt{F}, w) = \max\{6, 3, 4\} = 6$$

which finds the most probable assignment. Consider the semiring $(\mathbb{N}, +, \times, 0, 1)$ instead. Then:

$$\#(\mathtt{F}, w) = 6 + 3 + 4 = 13$$

which gives us the weighted model count.

Example 13. Extending the discussion in [55], we consider a class of mathematical programs where linear constraints and propositional formulas can be combined freely. See Figure 3 for an example with non-linear objectives. Formally, quantifier-free linear integer arithmetic and propositional logic are specified as the underlying logical systems, and the domains of constants are typed. The program declares formulas `F`, `G`, and `H`.

The counting task is non-factorized, and our convention for assigning weights to models is by letting the

`declare-weight`

directive also take arbitrary formulas as arguments. Of course, `TRUE` holds in every model, and so, the weight of every model is determined by the evaluation of `x1 * x2` at the model, that is, for any M, its weight is $\star^M(\mathtt{x1}^M, \mathtt{x2}^M)$. For example, a model that assigns `1` to `x1` and `1` to `x2` is accorded the weight `1 * 1`. Computing the count over $(\mathbb{N}, \max, 0)$ then yields a model of `H` with the highest value for `x1 * x2`.

To see this program in action, consider that every model of `H` must satisfy `p2`, and so must admit `3*x1 <= 4` and `2*x2 <= 5`. Since `{x1,x2}` can only take values `{1, ...,10}`, given the constraints, the desired model must assign `x1` and `x2` to `1` and `2` respectively. Then, its weight is `1*2`.

Encoding finite domain constraint satisfaction problems as propositional satisfiability is well-known. The benefit, then, of appealing to our framework is the ability to easily formulate counting instances:

Proposition 14. *Suppose Q is a CSP over variables $\mathcal{X}$, domains $\mathcal{D}$ and constraints C. There is a first-order theory $(\mathcal{L}, \mathcal{M}, \rhd)$, a $\mathcal{L}$-sentence ϕ and a weight function w such that $\#(\phi, w) = 1$ over $(\{0, 1\}, \vee, 0)$ iff Q has a solution. Furthermore, Q has n solutions iff $\#(\phi, w) = n$ over $(\mathbb{N}, +, 0)$.*

```
(set-logic FOL)
(set-algebra [NAT,+,0])
(set-type COLOR={r,b,g})
(set-type NODE={1,2,3})
(declare-predicate node (NODE))
(declare-predicate edge (NODE,NODE))
(declare-predicate color (NODE,COLOR))
N = (node(1) and node(2) and node(3))
E = (edge(1,2) and edge(2,3) and edge(3,1))
DATA = (N and E)
CONS = /* coloring constraints (omitted) */
(declare-weight TRUE 1)
(count (DATA and CONS))
```

Figure 4: counting graph coloring instances

Constraints are Boolean-valued functions [42], and so constraints over $\mathcal{X}$ can be encoded as $\mathcal{L}$-sentences, as in the example below:

Example 15. See Figure 4 for a counting instance of graph coloring: `edge(x,y)` determines there is an edge between `x` and `y`, `node(x)` says that `x` is a node, and `color(x,y)` says that node `x` is assigned the color `y`. The actual graph is provided using the formula `DATA`, which declares a fully connected 3-node graph. Also, `CONS` is a conjunction of the usual coloring constraints, *e.g.,* an edge between nodes `x` and `y` means that they cannot be assigned the same color.

Let $\mathcal{M}$ be a set of first-order structures for the vocabulary {`edge, node, color`}, respecting types from Figure 4. The interpretation of {`edge`, `node`} is assumed to be the same for all the models in $\mathcal{M}$ and is as given by `DATA`. Basically, then, the models differ in their interpretation of `color`. One model of ϕ = `(DATA and CONS)`, for example, is {`color(1,r), color(2,g), color(3,b)`}. The weights of all models is 1, and so, for $(\mathbb{N}, +, 0)$ we get:[2]

$$\#(\phi, w) = 6.$$

To summarize, the following result is easily shown for semiring programs:[3]

Proposition 16. *Suppose $\theta \in$ {SAT, #SAT, WMC, MPE, CSP, #CSP}. Suppose θ° is a solution to θ, in that $\theta^\circ \in \{0, 1\}$ for* SAT *and* CSP*, and $\theta^\circ \in \mathbb{R}$ for the rest. Then for any θ, there is a $\mathcal{T} = (\mathcal{L}, \mathcal{M}, \rhd)$, $\mathcal{S}$, w and $\phi \in \mathcal{L}$ such that $\#(\phi, w) = \theta^\circ$.*

3.2. Non-standard

A particular advantage of defining a logical theory in the way we did is that the framework is immediately applicable to model-level operations with non-standard semantics. We give a notable example that is simple to capture but which to the best of our knowledge has not been previously formulated in a semiring framework like ours.

Definition 17. A *stable model environment* is defined as follows. Let $\mathcal{T} = (\mathcal{L}, \mathcal{M}, \rhd)$ be a logical theory, where $\mathcal{L}$ is defined over a set of propositions $\mathcal{P}$ and $\mathcal{M}$ is the set of mappings from $\mathcal{P}$ to $\{0, 1\}$. Let us split $\mathcal{P}$ into two disjoint sets of variables founded variables $\mathcal{P}_f$ and standard variables $\mathcal{P}_s$. An answer set program δ is a tuple $(\mathcal{P}, \mathcal{R}, C)$ where $\mathcal{R}$ is a set of rules of form: $a \leftarrow b_1 \wedge \ldots \wedge b_n \wedge \neg c_1 \wedge \ldots \wedge \neg c_m$ such that $a \in \mathcal{P}_f$, the body variables $\{b_1, \ldots, c_m\} \subseteq \mathcal{P}$, and C

[2] In an analogous fashion, soft and weighted CSPs can be expressed using semirings [18].

[3] We remark that although the count operator in itself does not provide the variable assignments for SAT, MPE and CSP, we assume this can be retrieved from the satisfying interpretations.

is a set of constraints over the propositions (specified as rules with an empty head). A rule is positive if its body only contains positive founded literals. The least assignment of a set of positive rules $\mathcal{R}$, written $L(\mathcal{R})$ is the one that that satisfies all the rules and contains the least number of positive literals. Given an assignment $M \in \mathcal{M}$ and a program δ, the reduct of M wrt δ, written, δ^M is a set of positive rules that is obtained as follows: for every rule r, if any $c_i \in M$, or $\neg b_j \in M$ for any standard positive literal, then r is discarded, otherwise, all negative literals and standard variables are removed from r and it is included in the reduct. An assignment $M \in \mathcal{M}$ is a stable model of a program δ iff it satisfies all its constraints and M *agrees* with $L(\delta^M)$ wrt $\mathcal{P}_f$. We say two assignments M and M' agree wrt $\mathcal{P}'$ iff the set of positive literals (restricted to $\mathcal{P}'$) in M is identical to the set in M', and the set of negative literals (restricted to $\mathcal{P}'$) in M is identical to the set in M'. For a program δ, we let $M \rhd \phi$ iff M is a stable model of δ.

This is basically an adaptation of the answer set programming (ASP) by SAT formulation in [4]. By way of the semirings considered in Example 11, it immediately follows that:

Proposition 18. *Suppose $\mathcal{T} = (\mathcal{L}, \mathcal{M}, \rhd)$ is a stable model environment, $\delta \in \mathcal{L}$ is a program (as defined above). Suppose $\mathcal{S}$ is the semiring $(\{0,1\}, \vee, 0)$ and $w(M) = 1$ for every $M \in \mathcal{M}$. Then $\#(\delta, w)$ tells us whether there is a stable model. For the semiring $(\mathbb{N}, +, 0)$, $\#(\delta, w)$ yields the number of stable models.*

This formulation semantically characterizes the task of finding stable models as well as stable model counting. As with classical model counting [5], algorithmic solutions for the task may involve further optimizations to cache intermediate computations, among other ideas [4].

Example 19. Let us consider an example of a non-standard, non-factorized counting problem. Here, we will attempt to capture a simple formulation of inductive logic programming (ILP) [75]. Consider a logical language $\mathcal{L}$ with predicates $\{P(x), \ldots, Q(x,y), \ldots, R(x,y,\ldots,z)\}$ and finitely many constants. Consider the background theory $\mathcal{B}$, which is a conjunction of definite clauses defined over these predicates, corresponding to a logic program, as well as a set $\mathcal{E} = \{Q(a,b), \ldots, Q(c,d)\}$ of positive examples (ground atoms) for a target predicate $Q(x,y)$. Given a hypothesis space $\mathcal{H} = \{H_1, \ldots, H_n\}$, consider the formula $\mathcal{I} = \mathcal{B} \wedge (H_1 \underline{\vee} \ldots \underline{\vee} H_n)$, where $\phi_1 \underline{\vee} \ldots \underline{\vee} \phi_n$ means that exactly one of ϕ_i is true in a model. Recall that the standard formulation of inductive logic programming expects to guess a hypothesis H_i such that $\mathcal{B} \wedge H_i$ entails the examples. Here, for the sake of simplicity, we are assuming that we consider the set of hypothesis via an exclusive-or connective, which means that in any given model of $\mathcal{I}$, the model would satisfy $\mathcal{B}$ and exactly one of H_i. Thus, we are interested in $\mathcal{L}$-models of $\mathcal{I}$ that entail the examples.

For instance, given a background theory over domain $\{a, \ldots, d\}$ with atoms such as *parent(a,b)* and *parent(b,c)* in $\mathcal{B}$, and positive examples such as *grandparent(a,c)*, we are interested in models of the background theory that make *grandparent(x,z)* true whenever *parent(x,y)* and *parent(y,z)* are true. That is, we are interested in learning the clause: *grandparent(x,z)* $\leftarrow$ *parent(x,y), parent(y,z).*

In such a setup, we can define the weight w of a model M by:

$$|\{e \in \mathcal{E} \mid M \rhd e\}| - |\mathcal{E}|.$$

Then computing the count of $\mathcal{I}$ over $(\mathbb{N}, \min, 0)$, i.e., $\#(\mathcal{I}, w)$, yields a model where the hypothesis involving $Q(x,y)$ is such that the maximum number of positive examples are true. In fact, a count of 0 yields a model where the hypothesis fully captures the examples.

3.3. Infinite: Non-factorized

Defining measures [52] on the predicate calculus is central to logical characterizations of probability theory [53, 24]. We adapt this notion for semirings to introduce a general form of counting. For technical reasons, we assume that the universe of the semirings is $\mathbb{R}$. (If required, the range of the weight function can always be restricted to any subset of $\mathbb{R}$.)

Definition 20. Let $\mathcal{S} = (\mathbb{R}, \oplus, \mathbf{0})$ be any semiring and $\mathcal{T} = (\mathcal{L}, \mathcal{M}, \rhd)$ a theory. Let Σ be a σ-algebra over $\mathcal{M}$ in that $(\mathcal{M}, \Sigma)$ is a σ-finite measurable space wrt the measure $\mu \colon \Sigma \to \mathbb{R}$ respecting $\oplus$. That is, for all $E \in \Sigma$, $\mu(E) \geq 0$,

$\mu(\{\}) = 0$ and μ is closed under complement and countable unions: for all pairwise disjoint countable sets $E_1, \ldots \in \Sigma$, we have

$$\mu(\bigcup_{j=1}^{\infty} E_j) = \bigoplus_{j=1}^{\infty} \mu(E_j).$$

Moreover, because the spaces are σ-finite, $\mathcal{M}$ is the countable union of measurable sets with finite measure. Then for any $\phi \in \mathcal{L}$, $\#(\phi, \mu) = \mu(\mathcal{M}(\phi))$.

Example 21. We demonstrate that convex optimization can be cast as a semiring programming problem. Suppose $\mathcal{T} = (\mathcal{L}, \mathcal{M}, \rhd)$ is a first-order theory only containing constants $\mathcal{X} = \{x_1, \ldots, x_k\}$ with domains $D_i = \mathbb{R}$. Suppose $\phi(x_1, \ldots, x_k) \in \mathcal{L}$ is a conjunction of formulas of the form $c_1 x_1 + \ldots + c_k x_k \leq d$, for real numbers $c_1, \ldots, c_k, d$, and thus describing a polyhedron. In other words, for every $M \in \mathcal{M}$, $dom(M) = \mathbb{R}$, and so, M is a real-valued assignment to $\mathcal{X}$. In particular, if $M \rhd \phi$, then $x_1^M, \ldots, x_k^M$ is a point inside the polyhedron ϕ. Let Σ be a σ-algebra over $\mathcal{M}$ and so every $E \in \Sigma$ is a measurable set of points.

Suppose $f(x_1, \ldots, x_k) \colon \mathbb{R}^k \to \mathbb{R}$ is a convex function that we are to minimize. Consider the semiring $(\mathbb{R}, \inf, 0)$ and a measure μ such that for any $E \in \Sigma$:

$$\mu(E) = \inf \left\{ f(x_1^M, \ldots, x_k^M) \mid M \in E \right\}$$

which finds the infimum of the f-values across the assignments in E. Then, $\mu(\mathcal{M}(\phi)) = \#(\phi, \mu)$ gives the minimum of the convex function in the feasible region determined by ϕ.

Suppose f is a concave function that is to be maximized. We would then use $(\mathbb{R}, \sup, 0)$ instead, which finds the supremum of the f-values across assignments in $E \in \Sigma$.

More generally, the same construction is easily shown to be applicable for other families of mathematical programming (*e.g.,* non-linear optimization) as follows:

Proposition 22. *Suppose $\{x_1, \ldots, x_k\}$ is a set of real-valued variables. Suppose $P \subset \mathbb{R}^k$ is the feasible region of an optimization problem of the form $g_i(x_1, \ldots, x_k) \circ d_i$ for $i \in \mathbb{N}$, where $\circ \in \{\leq, <, \geq, >\}$ and $g_i \colon \mathbb{R}^k \to \mathbb{R}$. Suppose $f \colon \mathbb{R}^k \to \mathbb{R}$ is a function to be maximized (minimized). Then there is a $\mathcal{T} = (\mathcal{L}, \mathcal{M}, \rhd), \mathcal{S}, \mu$ and $\phi \in \mathcal{L}$ such that $\#(\phi, \mu)$ is the maximum (minimum) value for f in the feasible region P.*

Example 23. For a non-trivial example, consider the problem of matrix factorization, a fundamental concern in information retrieval and computer vision [30]. Given a matrix $I \in \mathbb{R}^{p \times n}$, we are to compute matrices $L \in \mathbb{R}^{p \times k}$ and $R \in \mathbb{R}^{n \times k}$, such that $e = \|I - LR^T\|$ is minimized. Here, $\| \cdot \|$ denotes the Frobenius Norm. Using real arithmetic, we provide a formulation in Figure 5. (Free variables are assumed to be implicitly quantified from the outside.)

Let $\mathcal{T} = (\mathcal{L}, \mathcal{M}, \rhd)$ be the theory of real arithmetic, where $\mathcal{L}$ includes the following function symbols:

$$\{\texttt{input, left, right, app, err}\}.$$

Here, `input(x,y)` is a real-valued function such that `x` $\in \{1, \ldots, p\}$ and `y` $\in \{1, \ldots, n\}$ in that `input(m,n)` is the entry at the m^{th} row and the n^{th} column of the matrix I; these entries are specified by `DATA`. Letting ϕ = (`DATA and F and G`), the set $\{M \in \mathcal{M} \mid M \rhd \phi\}$ are those $\mathcal{L}$-models whose interpretation of `input` is fixed by `DATA`. Basically, these models vary in their interpretations of $\{$`left, right`$\}$, which determines their interpretations for `app` and `err`. Here, `app` computes the product of the matrices `left` and `right`, and `err` computes the Frobenius Norm wrt `app` and `input`.

Let Σ be a σ-algebra over $\mathcal{M}$. The weight function in Figure 5 determines a measure μ such that for any $E \in \Sigma$:

$$\mu(E) = \inf \left\{ \texttt{err}^M \mid M \in E \right\}.$$

Therefore, $\#(\phi, \mu)$ yields the lowest `err` value; the model M such that $\texttt{err}^M = \#(\phi, \mu)$ is one with the best factorization of matrix I.

```
(set-logic LRA)
(set-algebra [REAL,inf,0])
(declare-function input (INT,INT) REAL)
...  /* declare left, right, app */
(declare-function err () REAL)
DATA = /* entries in input matrix (omitted) */
F = app(x,y) == sum{e} left(x,e)*right(e,y)
G = err == norm(sum{x,y} input(x,y) - app(x,y))
(declare-weight TRUE err)
(count (DATA and F and G))
```

Figure 5: matrix factorization

3.4. Infinite: Factorized

Despite the generality of the above definition, we would like to address the factorized setting for a number of applications, the most prominent being probabilistic inference in hybrid graphical models [13]. Consider, for example, a joint distribution on the probability space $\mathbb{R} \times \{0, 1\}$. Here, it is natural to define weights for each random variable separately, prompting a factorized formulation of counting. More generally, in many robotic applications, such hybrid spaces are common [87]. The main idea then is to apply our definition for counting by measures to each variable independently, and construct a measure for the entire space by *product measures* [52].

A second technicality is that in the finite case, the set of literals true at a model was finite by definition. This is no longer the case. For example, suppose x is a real-valued variable in a language $\mathcal{L}$, and M is a $\mathcal{L}$-model that assigns 3 to x. Then, $M \rhd (x = 3)$ but also $M \rhd (x \neq 3.1)$, $M \rhd (x \neq 3.11), \ldots$, and so on. Thus, for technical reasons, we assume that $\mathcal{L}$ only consists of constant symbols $\mathcal{X} = \{x_1, \ldots, x_k\}$ with fixed (possibly infinite) domains $\{D_1, \ldots, D_k\}$; the measures are defined for these domains.

Definition 24. Let $\mathcal{S} = (\mathbb{R}, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ be any semiring, $\mathcal{T} = (\mathcal{L}, \mathcal{M}, \rhd)$ any theory where $vocab(\mathcal{L}) = \{x_1, \ldots, x_k\}$ over fixed domains D_i. Suppose $\phi \in \mathcal{L}$. Suppose Σ_i is a σ-algebra over D_i in that (D_i, Σ_i) is a σ-finite measurable space wrt the measure $\mu_i \colon \Sigma_i \to \mathbb{R}$ respecting $\oplus$ (as in Definition 20). Define the product measure $\mu^* \doteq \mu_1 \times \cdots \times \mu_k$ on the measurable space $(D_1 \times \cdots \times D_k, \Sigma_1 \times \cdots \Sigma_k)$ satisfying:[4]

$$\mu^*(E_1 \times \cdots \times E_k) = \mu_1(E_1) \otimes \cdots \otimes \mu_k(E_k)$$

for all $E_i \in \Sigma_i$. Finally, define

$$\#(\phi, \mu^*) = \mu^*([\phi])$$

where $[\phi] = \left\{ x_1^M \times \ldots \times x_k^M \mid M \in \mathcal{M}(\phi) \right\}$.

Intuitively, $E_1 \in \Sigma_1, \ldots, E_k \in \Sigma_k$ capture sets of assignments, and the product measure considers the algebraic product of the weights on assignments to terms. As before, for $E_i, E_i' \in \Sigma_i$, $\mu_i(E_i \cup E_i') = \mu_i(E_i) \oplus \mu_i(E_i')$. Finally, precisely because the measures are defined on the domains of the terms, we obtain these for all of the satisfying interpretations using the construction $[\phi]$.

Example 25. We demonstrate the problem of finding the volume of a polyhedron, needed in the static analysis of probabilistic programs [81, 24]. Suppose $\mathcal{T} = (\mathcal{L}, \mathcal{M}, \rhd)$ and $\phi \in \mathcal{L}$ is as in Example 21, that is, ϕ defines a polyhedron. For every 0-ary function symbol $x_i \in \{x_1, \ldots, x_k\}$ with domain $D_i = \mathbb{R}$, let Σ_i be the set of all Borel subsets of $\mathbb{R}$, and let μ_i be the Lebesgue measure. Thus, for any $E \in \Sigma_i$, $\mu_i(E)$ gives the length of this line. Then, for the semiring $(\mathbb{R}, +, \times, 0, 1)$, the $+$ operator sums the lengths of lines for each variable, and $\times$ computes the products of these lengths. Thus, $\#(\phi, \mu^*)$ is the volume of ϕ.

[4] The product measure is unique owing to the σ-finite assumption via the Hahn-Kolmogorov theorem [52].

To see this in action, suppose $\phi = (2x \leq 5) \wedge (x \geq 1) \wedge (0 \leq y \leq 2)$. Then $\mathcal{M}(\phi) = \left\{\langle x \to n, y \to m\rangle \mid n, m \in \mathbb{R}, \triangleright\phi_{n,m}^{x,y}\right\}$, that is, all assignments to x and y such that $\phi_{n,m}^{x,y}$ is a valid expression in arithmetic. Therefore,

$$[\phi] = \{(n, m) \mid n \in [1, 2.5], m \in [0, 2], n \in \mathbb{R}, m \in \mathbb{R}\}.$$

Assuming μ_x is the Lebesgue measure for all Borel subsets of the domain of x, we have

$$\mu_x(\{n \mid n \in [1, 2.5], n \in \mathbb{R}\}) = 1.5.$$

Analogously, $\mu_y(\{m \mid m \in [0, 2], m \in \mathbb{R}\}) = 2$. Then, the volume is $\#(\phi, \mu^*) = \mu^*([\phi]) = 1.5 \times 2 = 3$.

Example 26. We demonstrate probabilistic inference in hybrid models [13] by extending Example 25. Consider a probabilistic program:

$X \sim$ UNIFORM$(0, 1)$
$Y \sim$ FLIP$(0, 1)$
if $(X > .6$ AND $Y \neq 1)$ **then** return DONE

In English: X is drawn uniformly from [0,1] and $Y \in \{0, 1\}$ is the outcome of a coin toss. If $X > .6$ and Y is not 1, the program terminates successfully. Suppose we are interested in the probability of DONE, which is expressed as the formula:

$$\phi = (0 \leq X \leq 1) \wedge (Y = 0 \vee Y = 1) \wedge (X > .6 \wedge Y \neq 1).$$

Suppose $\mathcal{T} = (\mathcal{L}, \mathcal{M}, \triangleright)$ is the theory of linear real arithmetic, with $vocab(\mathcal{L}) = \{X, Y\}$, $D_X = \mathbb{R}$ and $D_Y = \{0, 1\}$. As in Example 25, let Σ_X be the set of all Borel subsets of $\mathbb{R}$ and μ_X the Lebesgue measure. Let Σ_Y be the set of all subsets of $\{0, 1\}$ and μ_Y be the counting measure, *e.g.,* $\mu_Y(\{\}) = 0, \mu_Y(\{1\}) = 1$, and $\mu_Y(\{0, 1\}) = 2$. So $\mathcal{M}(\phi) = \left\{\langle X \to n, Y \to m\rangle \mid n \in \mathbb{R}, m \in \{0, 1\}, \triangleright\phi_{n,m}^{X,Y}\right\}$ and:

$$[\phi] = \{(n, m) \mid n > .6, 0 \leq n \leq 1, n \in \mathbb{R}, m \in \{0, 1\}\} \cap \{(n, m) \mid 0 \leq n \leq 1, n \in \mathbb{R}, m \neq 1, m \in \{0, 1\}\}.$$

This means that $\mu_X(\{n \mid \exists m \ (n, m) \in [\phi]\}) = .4$ and also $\mu_Y(\{m \mid \exists n \ (n, m) \in \phi\}) = 1$. Thus, $\mu^*([\phi]) = .4 \times 1$. Analogously, $\mu^*([\phi \vee \neg\phi]) = 2$. Therefore, the probability of DONE is $.4/2 = .2$.

In general, we have a variant of Proposition 22 [54]:

Proposition 27. *Suppose $\{x_1, \ldots, x_k\}$ is any set of real-valued variables. Suppose D is any countable set. Suppose $P \subset \mathbb{R}^m \times D^n$, where $m + n = k$, is any region given by conjunctions of expressions of the form $c_1 x_1 + \ldots + c_k x_k \circ e$ and $x_i \diamond d$, where $\circ \in \{\leq, <, \geq, >\}$, $\diamond \in \{\neq, =\}$, $c_i, e \in \mathbb{R}$ and $d \in D$. Then there is a $\mathcal{T} = (\mathcal{L}, \mathcal{M}, \triangleright), \mathcal{S}, \mu$ and $\phi \in \mathcal{L}$ such that $\#(\phi, \mu)$ is the volume of P.*

4. Towards Compositionality

A noteworthy feature of many logic-based knowledge representation formalisms is their compositional nature. In semiring programming, using the expressiveness of predicate logic, it is fairly straightforward to combine theories over possibly different signatures (*e.g.,* propositional logic and linear arithmetic), as seen, for example, in SMT solvers [9].

A more intricate flavor of compositionality is when the new specification becomes difficult (or impossible) to define using the original components. This is a common occurrence in large software repositories, and has received a lot of attention in the AI community [68].

In this section, we do not attempt to duplicate such efforts, but propose a different account of compositionality that is closer in spirit to semiring programming. It builds on similar ideas for CSPs [16], and is motivated by machine learning problems where learning (*i.e.,* optimization) and inference (*i.e.,* model counting) need to be addressed in tandem. More generally, the contribution here allows us to combine two semiring programs, possibly involving different semirings. For simplicity of presentation, we consider non-factorized and finite problems over distinct vocabularies.

Definition 28. Suppose $\mathcal{S}_1 = (\mathbb{S}_1, \oplus_1, \mathbf{0}_1)$ and $\mathcal{S}_2 = (\mathbb{S}_2, \oplus_2, \mathbf{0}_2)$ are any two semirings. We define their *composition* $\mathcal{S} = (\mathbb{S}, \oplus, \mathbf{0})$ as:

- $\mathbb{S} = \{(a, b) \mid a \in \mathbb{S}_1, b \in \mathbb{S}_2\}$;
- $\mathbf{0} = (\mathbf{0}_1, \mathbf{0}_2)$;
- for every $c = (a, b) \in \mathbb{S}$ and $c' = (a', b') \in \mathbb{S}$, let $c \oplus c' = (a \oplus_1 a', b \oplus_2 b')$.

That is, the composition of the semirings is formed from the Cartesian product, respecting the summation operator for the individual structures.

Definition 29. Suppose $\mathcal{T}_1 = (\mathcal{L}_1, \mathcal{M}_1, \rhd_1)$ and $\mathcal{T}_2 = (\mathcal{L}_2, \mathcal{M}_2, \rhd_2)$ are theories, where $\mathcal{L}_1$ and $\mathcal{L}_2$ do not share atoms, $\mathcal{S}_1$ and $\mathcal{S}_2$ semirings, and w_1 and w_2 weight functions for $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively. Given the environments $(\mathcal{T}_1, \mathcal{S}_1, w_1)$ and $(\mathcal{T}_2, \mathcal{S}_2, w_2)$, we define its *composition* as $(\mathcal{T}, \mathcal{S}, w)$, where $\mathcal{S}$ is a composition of $\mathcal{S}_1$ and $\mathcal{S}_2$ and $\mathcal{T} = (\mathcal{L}, \mathcal{M}, \rhd)$:

- $\phi \in \mathcal{L}$ is obtained over Boolean connectives from $\mathcal{L}_1 \cup \mathcal{L}_2$.
- $\mathcal{M} = \{(M_1, M_2) \mid M_i \in \mathcal{M}_i\}$.
- The meaning of $\phi \in \mathcal{L}$ is defined inductively:
  - $(M_1, M_2) \rhd p$ for atom p iff $M_1 \rhd_1 p$ if $p \in \mathcal{L}_1$ and $M_2 \rhd_2 p$ otherwise;
  - $(M_1, M_2) \rhd \neg\phi$ iff not the case that $(M_1, M_2) \rhd \phi$;
  - $(M_1, M_2) \rhd \phi_x \wedge \phi_y$ iff $(M_1, M_2) \rhd \phi_i$ for $i \in \{x, y\}$.
- For any $(M_1, M_2) \in \mathcal{M}$, $w((M_1, M_2)) = (w_1(M_1), w_2(M_2))$.

In essence, the Cartesian product for the semirings is extended for arbitrary theories and weight functions. The meaning of formulas rests on the property that $\mathcal{L}_1$ and $\mathcal{L}_2$ do not share atoms.[5] It is now easy to see that the counting for problem for $\mathcal{T}$ works as usual: that is, for any $\phi \in \mathcal{L}$,

$$\#(\phi, w) = \bigoplus_{(M_1, M_2) \rhd \phi} w((M_1, M_2)).$$

Example 30. We demonstrate a (simple) instance of combined learning and inference. Imagine a robot navigating a world by performing *move* actions, and believes its actuators need repairs. But before it alerts the technician, it would like to test this belief. A reasonable test, then, is to inspect its trajectory so far, and check whether the expected outcome of a move action in the current state matches the behavior of the very first move action. More precisely, the robot needs to appeal to linear regression to estimate its expected outcome, and query its beliefs based on the regression model. We proceed as follows.

Environment 1

Let $\mathcal{T}_1 = (\mathcal{L}_1, \mathcal{M}_1, \rhd_1)$ be the theory of real arithmetic with $vocab(\mathcal{L}_1) = \{s_0, s_1, \ldots, s_k, a, b, e\}$. Suppose that by performing a *move* action, the robot's position changes from s_i to s_{i+1}. Let $\phi_1 \in \mathcal{L}_1$ be as follows:

$$(s_0 = 1 \wedge s_1 = 2 \wedge s_2 = 3) \wedge$$
$$e = \textstyle\sum_i (s_{i+1} - b - a \cdot s_i)^2$$

The idea is that the values of s_i are the explanatory variables in the regression model and s_{i+1} are the response variables, that is, the trajectory data is of the form $\{(s_0, s_1), (s_1, s_2)\}$.

[5] This is analogous to SMT solvers for combinations of theories [9]. However, see [68, 34] for accounts of modularity based on first-order structures sharing vocabularies (and thus, atoms).

Consider a weight function $w_1 \colon M \to e^M$. That is, the weight of a model M is the universe element corresponding to the constant e in M, analogous to Figure 3. (For simplicity, we assume that the coefficients of the regression model are natural numbers.) For the semiring $\mathcal{S}_1 = (\mathbb{N}, \min, \infty)$:

$$\#(\phi_1, w_1) = \min\left\{e^M \mid M \rhd \phi_1, M \in \mathcal{M}_1\right\}.$$

In other words, models in $\mathcal{M}_1$ interpret s_i as given by the data, and models differ in their interpretation of a, b and thus, e. For the data in ϕ_1, we would have a model M where $e^M = 0$, $a^M = 1$ and $b^M = 1$, and so $\#(\phi_1, w_1) = 0$.

Environment 2

Let $\mathcal{T}_2 = (\mathcal{L}_2, \mathcal{M}_2, \rhd_2)$ be a propositional theory, where $vocab(\mathcal{L}_2) = \{repair\}$. Imagine a weight function w_2 as follows: $w_2(\{repair\}) = .7$, and $w_2(\{\neg repair\}) = .3$. That is, the robot believes that repairs are needed with a higher probability. Indeed, for the semiring $(\mathbb{R}, +, 0)$:

$$\#(repair, w_2) = .7 \text{ versus } \#(\neg repair, w_2) = .3.$$

Composition

Let $(\mathcal{T}, \mathcal{S}, w)$ be the composition of the two environments with $\mathcal{T} = (\mathcal{L}, \mathcal{M}, \rhd)$ and $\mathcal{S} = (\mathbb{S}, \oplus, \mathbf{0})$. Suppose $\phi \in \mathcal{L}$ is as follows:

$$\begin{array}{l}\phi_1 \wedge (s_3 = a \cdot s_2 + b) \wedge \\ (s_3 - s_2) \neq (s_1 - s_0) \equiv repair.\end{array}$$

The final conjunct basically checks whether the expected change in position matches what was happening initially, and if not, *repair* should be true.

To see Definition 29 in action, observe that for any $M_1 \in \mathcal{M}_1, M_2 \in \mathcal{M}_2$, $(M_1, M_2) \in \mathcal{M}$, the formulas ϕ_1 and those involving s_i are interpreted in M_1, but *repair* in M_2. The weight function is as follows. Given $(M_1, M_2) \in \mathcal{M}$ and $(M_1', M_2') \in \mathcal{M}$, we have:

$$\begin{array}{l} w((M_1, M_2)) \oplus w((M_1', M_2')) \\ \quad = (\ \min(w_1(M_1), w_1(M_1')),\ w_2(M_2) + w_2(M_2')\ ). \end{array}$$

Then the robot can obtain the weight of *repair* and ϕ using:

$$\#(\phi \wedge repair, w) = (0, 0)$$

where, of course, the first argument is the error of the regression model and the second is 0 because $\phi \wedge repair$ is inconsistent. That can be contrasted to the count below:

$$\#(\phi \wedge \neg repair, w) = (0, .3).$$

It is also easy to see that $\#(\phi, w) = (0, .3)$.

As in WMC [23], suppose the robot obtains the probability of a query q given ϕ using:

$$\frac{\#(\phi \wedge q, w)}{\#(\phi, w)}$$

where the division is carried out by ignoring the regression error. Then the probability of $\neg repair$ given ϕ is 1. Thus, no repairs are needed.

Example 31. We show that by combining theories, a natural semantics can be given to hybrid problems such as task and motion planning. Here, the concern is to integrate a symbolic high-level planner, described in logic, and geometric constraints; see [84] for terminology and notation. Suppose $\mathcal{Z}$ is the theory of integer arithmetic that interprets a motion planning space (C, N, p) where $C = \mathbb{Z}$ is the configuration space, $N \subset C$ is an obstacle region, and $p \in C$ is the initial pose of a robot's gripper. (For simplicity, we consider a one-dimensional setting.) The result of

doing a motion t is the pose $p + t$. We axiomatize that x is reachable from y by following t without touching obstacles as:

$$\textit{IsReachable(x,y,t)} \equiv (x = y + t) \wedge \forall z(z \leq t \supset (y + z \notin N)).$$

Next, suppose $\mathcal{T}$ is a propositional theory that interprets a task planning space (S, A, i, g), where A is a set of actions, S a set of states, and $i, g \in S$ are the initial and goal states. The task is to synthesize action sequences reaching g. However, in robotic applications, actions are predicated on geometric constraints; *e.g.,* the precondition for *pickup(o,p,t)* – the action of picking up an object o wrt a trajectory t and the gripper's pose p – is the sentence:

$$\textit{IsReachable(o,p,t)} \wedge \textit{IsGripperFree}.$$

Interestingly, *IsGripperFree* is from the language of $\mathcal{T}$ but *IsReachable(x,y,z)* is from that of $\mathcal{Z}$, and using our semantic setup, such complex systems can be easily interpreted.

To describe an illustrative counting problem, suppose that for every $\mathcal{Z}$-model M and $\mathcal{T}$-model M', (M, M') determines a combined task-motion plan of some length. A plan is valid iff g can be reached from i by following this plan wrt the domain's axioms (*e.g.,* avoiding obstacle regions and satisfying action preconditions). Suppose plans are of the form $t_1, a_1, \ldots, t_m, a_n$, where $a_i \in A$ and t_j are motions. Suppose actions and motions incur costs, and the weight of a model is $\sum_i cost(a_i) + \sum_j cost(t_j)$ wrt the plan it determines. Assuming $\mathcal{Z} \cup \mathcal{T}$ is a finite theory (for simplicity), the semiring $(\mathbb{R}, \min, +, 0, 1)$ yields a cost optimal plan.[6]

5. Discussion & Challenges

5.1. Solver Strategy

The upshot of semiring programming is that it encourages us to inspect strategies for an inferential mechanism [57]. This has to be done carefully, as we would like to build on scalable methodologies in the literature, by restricting logical theories where necessary. In this section, we discuss whether our programming model can be made to work well in practice. We remark at the outset that while there are many benefits in viewing numerous challenging problems as instances of one framework, implementing a solver strategy is likely to be challenging. The purpose of the implementation would not be to improve on the solving on individual counting instances, but rather encourage us to look at strategies for effectively solving combinations and compositions of counting instances both in discrete as well as continuous domains.

Let us consider two extremes, and then a hybrid option:

- **Option 1:** At one extreme is a solver strategy based on a single computational technique. Probabilistic programming languages, such as Church [48], have made significant progress in that respect for generative stochastic processes by appealing to Markov Chain Monte Carlo sampling techniques. Unfortunately, such sampling techniques do not scale well on large problems and have little support for linear and logical constraints.
- **Option 2:** At the other extreme is a solver strategy that is arbitrarily heterogeneous, where we develop unique solvers for specific environments, that is, $(\mathcal{T}, \mathcal{S})$ pairs.
- **Option 3:** We believe the most interesting option is in between these two extremes. In other words, to identify the smallest set of computational techniques, and effectively integrate them is both challenging and insightful. This may mean that such a strategy is less optimal than Option 2 for the environment, but we would obtain a simpler and more compact execution model. To that end, let us make the following observations from our inventory of examples:
  - **Finite versus infinite:** variable assignments are taken from finite sets versus infinite or uncountable sets.
  - **Non-factorized versus factorized:** the former is usually an optimization problem with an objective function that is to be maximized or minimized. The latter is usually a counting problem that can be obtained locally and concatenated for a global solution.

[6] Prior work on semiring composition [16] can further allow us to minimize one aspect (*e.g.,* cost) and maximize another (*e.g.,* number of rooms cleaned).

– **Compositionality:** locally consistent solutions (*i.e.,* in each environment $(\mathcal{T}, \mathcal{S})$) need to be tested iteratively for global consistency.

– **Exact versus approximate:** A distinction is also worth making between exact and approximate solvers. In the literature on WMC, for example, there are both exact and approximate solvers [46], and in probabilistic programming as well, we might consider approaches based on knowledge compilation [26] and those based on sampling [48, 77]. In general, approximation may be unavoidable in the infinite domain setting, and moreover, it might achieve better overall computational properties than exact solutions to subproblems.

  In particular, although much of the discussion below is motivated by WMC and its extensions (and so keeping in line with AMC), approximation algorithms such as message passing [73] and minibucket elimination [27], which are also closely related to WMC, may be promising candidates too. Approximation strategies such as those discussed in [66] hint at how perhaps sub-computations for problems can be unified for an effective global approximation.

Thus, Option 3 could be realized as follows:

– Factorized problems need a methodology for effective enumeration, and therefore, advances in model counting [46], such as knowledge compilation, are the most relevant. For finite theories, we take our cue from the ProbLog family of languages [38, 58], that have effectively applied arithmetic circuits for tasks such as WMC and MPE. In particular, it is shown in [59] how arbitrary semiring labels can be propagated in the circuit. See [36, 37] for progress on knowledge compilation in CSP-like environments.

  For infinite theories, there is growing interest in effective model counting for linear arithmetic using SMT technology [24, 13, 14]. For example, the formulation of weighted model integration [13, 14] generalizes the notion of weighted model counting from propositional logic to quantifier-free linear arithmetic. Roughly speaking, it amounts to computing the computing the volume of polytopes against a density function. Although limited to factorized problems over probability measures, recent work [61, 60] suggests compilation targets for such problems, which would be a promising start for generalizations to arbitrary semirings.

  When considering relational factorized problems, considerable attention has been paid to exploiting symmetries over indistinguishable sets of objects, e.g., [28]. It is not immediate whether these insights would also carry over to non-factorized problems, but that is worthwhile to consider. The case of attempting to lump these symmetries when considering compositions is also an interesting direction, regardless of whether this composition is over the same logical setup, or the cartesian composition developed above.

– For non-factorized problems, a natural candidate for handling semirings does not immediately present itself, making this is a worthwhile research direction.[7] Appealing to off-the-shelf optimization software [2] is always an option, but they embody diverse techniques and the absence of a simple high-level solver strategy makes adapting them for our purposes less obvious. In that regard, solvers for *optimization modulo theories* (OMT) [83] are perhaps the most promising. OMT technology extends SMT technology in additionally including a cost function that is be maximized (minimized). In terms of expressiveness, CSPs [76] and certain classes of mathematical programs can be expressed, even in the presence of logical connectives. In terms of a solver strategy, they use binary search in tandem with lower and upper bounds to find the maximum (minimum). This is not unlike DPLL traces in knowledge compilation, which makes that technology the most accessible for propagating arbitrary semiring labels.

– Compositional settings are, of course, more intricate. Along with OMT, and classical iterative methods like *expectation maximization* [62], there are a number of recent approaches employing branch-and-bound search strategies to navigate between local and global consistency [43]. Which of these can be made amenable to compositions of SP programs remains to be seen however.

[7] Approaches like [15, 37] on knowledge compilation for optimization and [82] on circuits for linear constraints are compelling, but their applicability to non-trivial arithmetic and optimization problems remains to be explored.

Overall, we believe the most promising first step is to limit the vocabulary of the logical language to propositions and constants (*i.e.,* 0-ary functions), which make appealing to knowledge compilation and OMT technology straightforward. It will also help us better characterize the complexity of the problems that SP attempts to solve.[8] At first glance, SP is seen to naturally capture #P-complete problems in the factorized setting, both in the finite case [46] and the infinite one [31]. In the non-factorized setting, many results from OMT and mathematical programming are inherited depending on the nature of the objective function and the domains of the program variables [56, 9, 83]. By restricting the language as suggested, the applicability of these results can be explored more thoroughly. Incidentally, such a language might also enable easier use of approximation techniques such as sampling and heuristics [48, 77, 43].

5.2. Non-standard problems

We argued that as far as expressiveness was concerned, our formal setup allowed us to capture non-standard problems (e.g., non-classical entailment). We demonstrated this in Section 3.2. Let us now make some observations about how this generality could impact solver strategies.

Consider that SP defines the semantical foundations for capturing numerous problems, but as the current section makes clear, the underlying computational strategy need not reflect the semantic definition in a strict fashion: for example, the most effective way to model count is not by enumerating every model in a sequential fashion, and likewise, evaluating the value of models (via an objective function) in a sequential fashion is simply not feasible in a continuous setting. Be that as it may, it is useful to identify the common computational underpinnings of the ASP and the ILP formulations from Section 3.2.

In the case of ASP, the SP formulation can be mapped to a SAT problem [4]. Given ASP's popularity for declaratively modeling challenging problems [20], and its recent extensions for handling objective functions and arithmetic constraints [69, 70], it is conceivable that a propositional model counting strategy for ASP might easily implement a reasonable set of SP formulations.

Interestingly, it is also known that the induction of clauses in ILP can be reduced to SAT [35]. (The approach in [35] differs somewhat from our formulation, but as discussed, our formulation is purely for semantical purposes.) This provides further evidence that a common computational strategy might exist for a reasonable set of standard and non-standard SP formulations: in particular, those that employ propositional model theory.

However, these two positive developments certainly do not capture the entire gamut of possibilities that can be instantiated with our setup. Moreover, even if (say) SAT reductions can be provided, it is possible that the development of a generic solver is awkward for all of the envisioned tasks. Thus, in general, it seems unlikely that a solver tailored for a certain non-standard formulation will immediately lead itself as a generic solution to a large class of SP formulations. A broader point here is that although capturing non-standard problems in SP is possible from a semantical viewpoint, it is not clear that a generic solver can be established that handles all such formulations. Indeed, for non-standard (but also, of course, standard) problems, it would be worthwhile to obtain a precise characterization of when it is needed to go beyond solvers that are generic in semirings but specific to theories. (Consider, for example, our discussion on Option 3.) Clearly, a SAT reduction might mean a common computational strategy at least for discrete settings, and a SMT reduction might mean a common computational strategy for discrete and continuous settings, provided we are in a domain with finitely many variables.

5.3. Composition

In Section 4, we argued for the usefulness of an operator that constructs a simple cartesian composition. That is, given $\mathcal{S}_1 = (\mathbb{S}_1, \oplus_1, \otimes_1, \mathbf{0}_1, \mathbf{1}_1)$ and $\mathcal{S}_2 = (\mathbb{S}_2, \oplus_2, \otimes_2, \mathbf{0}_2, \mathbf{1}_2)$, we define $\mathcal{S} = (\mathbb{S}, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ such that:

- $\mathbb{S} = \{(a, b) \mid a \in \mathbb{S}_1, b \in \mathbb{S}_2\}$,
- $\mathbf{0} = (\mathbf{0}_1, \mathbf{0}_2)$,
- $\mathbf{1} = (\mathbf{1}_1, \mathbf{1}_2)$, and
- for every $c = (a, b) \in \mathbb{S}, c' = (a', b') \in \mathbb{S}$, we let $c \oplus c' = (a \oplus_1 a', b \oplus_2 b')$ and $c \otimes c' = (a \otimes_1 a', b \otimes_2 b')$.

[8] Richer fragments have to be considered carefully to avoid undecidable properties [19].

Note that we have generalized the construction to also account for the multiplicative operator.

As discussed, the underlying paradigm is to segment computational problems in terms of counting tasks defined over a logical theory, a semiring, and compose programs using a suitable composition operator. While logical languages already support the combination of logical theories, a cartesian composition is also suggested above. But, of course, others maybe considered too. For example, in [67], the following composition operator for real fields is discussed in the context of gradients and other quantities: given $\mathcal{S}_1 = (\mathbb{R}_{\geq 0}, +, \times, 0, 1)$ and $\mathcal{S}_2 = (\mathbb{R}_{\geq 0}, +, \times, 0, 1)$, define $\mathcal{S} = (\mathbb{S}, \otimes, \oplus, \mathbf{0}, \mathbf{1})$ such that:

- $\mathbb{S} = \{(a, b) \mid a \in \mathbb{R}_{\geq 0}, b \in \mathbb{R}_{\geq 0}\}$,
- $\mathbf{0} = (0, 0)$,
- $\mathbf{1} = (1, 0)$, and
- for every $c = (a, b) \in \mathbb{S}, c' = (a', b') \in \mathbb{S}$, we let $c \oplus c' = (a + a', b + b')$ and $c \otimes c' = (a \times a', a \times b' + a' \times b)$.

Analogously, in very recent work [29], for example, the gradient semiring is considered, which is closely related to the expectation semiring above and has applications for learning utilities in decision-theoretic models. The discussion therein also hints that with some semirings, the order of the operations may need to be taken into account, which raises an interesting challenge for composition and solver construction. Thus, an interesting question for the future would be to identify the set of all such useful operators, the compositions thereof, and more importantly, investigate whether there is a core set of "axiomatic" or basic operators from which all other operators can be defined.

As discussed in the next section, there are also various proposals for defining a semantics for the communication of arbitrarily heterogenous computational modules [85]. It would be interesting to see whether SP could be further extended along those lines.

5.4. Interface

To reiterate a point made previously, the thrust of the proposal is to rigorously generalize standard AMC. To that end, a theoretical study on the semantics of programs is largely studied. However, we also illustrated the framework using a programming language inspired by SMT syntax, but this is meant to be a prototypical starting point. As this language allows a somewhat literal description of the underlying counting problem, we view it as a reasonable starting point. It is only one possible realization of the underlying semantics, and in that regard, it is not necessary that the actual programming interface resemble this syntax: users may feel more comfortable using a conventional imperative programming interface, or a functional one. As we discuss below, frameworks such as [17] that combine constraint logic programming and semirings realize a reasonable subset of the problems that SP can capture. (For example, they fall short of capturing tasks such as probabilistic inference in hybrid domains.) For that subset, the framework can be seen to provide a declarative logic-based programming interface.

Thus, with SP providing the underlying semantics, depending on the chosen programming paradigm (imperative, functional, or logic-based), it is conceivable that a powerful interface could be designed that would reduce to well-known frameworks for the corresponding SP problems. What is less clear, however, is whether that same interface would also allow us to express and solve SP tasks outside the standard problem types of those well-known frameworks.

6. Related Work

Semiring programming is related to efforts from different disciplines within AI, and we discuss representative camps. In a nutshell, SP can be seen as a very general semantical framework, as noted in Figure 6.

6.1. Statistical modeling

Formal languages for generative stochastic processes, such as Church [48] and BLOG [72], have received a lot of attention in the learning community. Such languages provide mechanisms to compactly specify complex probability distributions, and appeal to sampling for inference.

Closely related to such proposals are probabilistic logic programming languages such as Problog [26] that extends Prolog with probabilistic choices and uses WMC for inference [38]. In particular, a semiring generalization of Problog,

	FT	FN	IT	IN	C	S	R
APROBLOG	✓	×	×	×	✓	✓	✓•
AMC	✓	×	×	×	✓	✓	✓•
SPN	✓	×	✓	×	✓•	✓	×
CHURCH	✓	×	✓	×	✓•	×	×
ESSENCE	×	✓	×	×	✓•	×	×
CSP	×	✓•	×	✓•	✓•	×	×
SEMIRING CSP	×	✓•	×	×	✓•	✓	×
AMPL	×	✓	×	✓	✓•	×	×
ASP	✓	✓•	×	×	✓	×	✓
(O)SMT	×	✓	×	✓	✓	×	×
#SMT	✓	×	✓	×	✓	×	×
SP	✓	✓	✓	✓	✓	✓	✓

Figure 6: A comparison, where F = finite, T = factorized, N = non-factorized, I = infinite, C = logical connectives and quantifiers, R = non-standard, S = semiring apparatus, and • denotes that a feature is available in a restricted sense.

called aProbLog [58], was the starting point for our work and employs the semiring variation of WMC for inference [59]. A recent extension of aProbLog, called kProbLog, by [78] is able to further combine several semirings and does not require factorized weights as it uses meta-functions $w(a) = f(w(a_1), \ldots, w(a_n))$ to compute the weight $w(a)$ of an atom from the weights of the atoms $a_1, \ldots, a_n$ appearing in its proofs. But the kind of weight function and factorization used in kProbLog differs from the unfactorized weight function over the models used in the present paper. Nevertheless, kProbLog is able to represent tensors, compute kernel functions and perform algorithmic differentiation. It will be interesting to investigate whether kProbLog can be combined with semiring programming.

As discussed before, when considering discrete domains (but see below), SP is related to algebraic model counting (AMC) [59] and a semiring extension of sum-product networks (SPNs) [44]. Semantically, SP generalizes AMC and SPNs, in that SP's definitions for finite and factorized problems coincides with AMC and SPNs. Computationally, AMC and SPNs implement the sums of products of factors over discrete variables [5] via circuits (from knowledge compilation [25]). On a related note, a variable elimination strategy is considered in [1], where a variant of dynamic programming is proposed.

The differences between SP, on the one hand, and AMC and SPNs, on the other, are worth discussing further. As mentioned above, both AMC and SPNs argue for a circuit representation to capture the computational problem in discrete domains, which rests on a factorization of the weight function. As also discussed, in limited settings, continuous random variables can mapped to atomic propositions [44]. Alternatively, by interpreting the intervals of a continuous variable as propositions, propositional solvers can be leveraged for inference [14]. Such schemes may indeed allow a class of continuous SP problems to be approached by knowledge compilation. Intuitively, the key idea in both schemes is to abstract continuous features as propositional variables. However, in precisely this sense, they are not sufficiently general for SP. For one thing, even in the case of optimization problems, the compilation approach is appropriate only when the objective function is decomposable [5, 44]. For another, both the schemes are severely limited in terms of the expressiveness from an arithmetic reasoning point of view. With SMT, for example, it becomes possible to express probabilistic and categorical constraints such as $x > y + 5$ and $x > y \cdot z$; these are awkward or even impossible to capture using propositional abstractions. One other possibility is to use a finite-precision propositional encoding of continuous properties [9], but this may lead to an exponential blow-up in the theory and it is also unclear how effective the resulting circuit representation would be. (That is, even though circuit representations admit reasoning in time polynomial in size of the circuit, the circuit may be very large.)

All of this suggests that relying on the propositional model theory and weight factorizations might be too restrictive for our purposes. In many discrete cases, indeed, AMC and related techniques already offer a algorithmic realization of SP. But in other instances, we will want to rely on SMT, approximation strategies, and other possibilities discussed with Option 3 above. By providing the semantic mapping for a large class of problems, SP encourages us to think beyond existing (and very popular) discrete, factorized settings already addressed by AMC and SPNs.

In sum, on the one hand, these prior formulations do not permit arbitrary arithmetic reasoning, non-standard semantics, non-factorized weights and composition. Unfortunately, on the other, by giving up the propositional lan-

guage, and in particular, the use of arithmetic circuits, we do not enjoy the tractability results and unified evaluation scheme offered by AMC and SPNs. This is not surprising given the generality. Nonetheless, since SP semantically generalizes AMC, we can recover those advantages and appeal to AMC and SPNs if the domain is limited in the appropriate way. Consider also that a formulation in terms of sums of products of factors over discrete variables [5] was enormously useful to investigate and compare algorithms, and pave the way for strategies used in circuit compilation, and thus, AMC and SPNs. By generalizing the thrust of sums of products of factors to a continuous and compositional setting, SP is providing a semantics for a more general language. We hope that this might also lead to new solver strategies.

It is worth noting that all of these solver discussions primarily pertain to developments with finitely many variables, even if the values of these variables may be drawn from countably infinite or uncountably infinite sets. In a first-order setting, of course, we may have infinitely many variables, and first-order automated reasoning techniques might become useful. For example, the area of lifted reasoning [28] considers the challenge of exploiting symmetries to speed up inference in relational representations; however, the underlying assumption is that the domain is still finite. The case of handling infinite-variable first-order representations in a finitary manner is only understood in some special cases, e.g., [10]. See [11] for a review of the developments.

Finally, let us note that the use of semirings in machine learning is not new; languages such as aProbLog *e.g.,* [47] and Dyna [32] are motivated similarly to AMC. There are some differences, however: Dyna is based on Datalog; our logical setting is strictly more expressive than Datalog and its extensions (*e.g.,* non-Horn fragment, constraints over reals). Dyna also labels proofs but not interpretations, as would SP (thus capturing weighted model counting, for example). In addition to these accounts, numerous solvers that manipulate algebraic structures with numeric and symbolic entities have emerged in the recent years; see, for example, [77] and references therein.

6.2. Constraints

The constraints literature boasts a variety of modeling languages, such as Essence [45], among others [71, 88]. (See [39], for example, for a proposal on combining heterogeneous solvers.) On the one hand, SP is more expressive from a logical viewpoint as constraints can be described using arbitrary formulas from predicate logic, and we address many problems beyond constraints, such as probabilistic reasoning. On the other hand, such constraint languages make it easier for non-experts to specify problems while SP, in its current form, assumes a background in logic. Such languages, then, would be of interest for extending SP's modeling features.

A notable line of CSP research is by Bistarelli [16] and his colleagues [18, 16]. Here, semirings are used for diverse CSP specifications, which has also been realized in a CLP framework [17]. In particular, our account of compositionality is influenced by [16]. Under some representational assumptions, SP and such accounts are related, but as noted, SP can formulate problems such as probabilistic inference in hybrid domains that does not have an obvious analogue in these accounts.

6.3. Optimization

Closely related to the constraints literature are the techniques embodied in mathematical programming more generally. There are three major traditions in this literature that are related to SP. Modeling languages such as AMPL [40] are fairly close to constraint modeling languages, and even allow parametrized constraints, which are ground at the time of search. The field of disciplined programming [50] supports features such as object-oriented constraints. Finally, relational mathematical programming [3] attempts to exploit symmetries in parametrized constraints.

From a solver construction perspective, these languages present interesting possibilities. From a framework point of view, however, there is little support for logical reasoning in a general way.

6.4. Knowledge representation

Declarative problem solving is a focus of many proposals, including ASP [21], model expansion [74, 86], among others [22]. These proposals are (mostly) for problems in NP, and so do not capture #P-hard problems like model counting and WMC. Indeed, the most glaring difference is the absence of weight functions over possible worlds, which is central to the formulation of statistical models. Weighted extensions of these formalisms, *e.g.,* [6, 69], are thus closer in spirit.

The generality of SP also allows us to instantiate many such proposals, including formalisms using linear arithmetic fragments [13, 83]. Consider OMT for example. OMT can be used to express quantifier-free linear arithmetic sentences with a linear cost function, and a first-order structure that minimizes the cost function is sought. From a specification point of view, SP does not limit the logical language, does not require that objective functions be linear, and a variety of model comparisons, including counting, are possible via semirings. Compositionality in SP, moreover, goes quite beyond this technology.

There is also a longstanding interest in combining different (logical) search problems in a single logical framework, as seen, for example, in modular and multi-context systems [68, 34]. In such frameworks, it would be possible to get a graph coloring program and ASP program to communicate their solutions, often by sharing atoms. The semantics of such frameworks is defined in terms of a logical interpretation, such that its partial models capture sub-problems. The main differences to what has been attempted in this paper, however, is that the focus is purely on finding assignments to sub-problems, so it does not cover model counting and its extensions, such as model integration. Correspondingly, the notion of compositionality put forward in this paper is of a different nature.

On the one hand, it would be interesting to see whether modular systems can address problems such as combined inference and learning. On the other hand, it would worth considering whether SP captures all the problem types discussed in these other frameworks, and if not, whether SP could be extended further by borrowing ideas from modular systems.

In that spirit, the need for declarative programming for problem solving in AI has received considerable attention over the years. Recently, [63], for example, argue that especially in the context of data-driven methods, declarative programming approaches will likely be very useful to tame the model-building process, and point to numerous different research attempts, many of which are discussed here. The goal of that work was not to put forward any concrete proposal language, but rather discuss challenges for future frameworks, including the need for composition and abstraction. On the composition front, they suggest the composition offered by probabilistic programs as an attractive option; SP goes considerably beyond that simple type of composition. Whether SP or a suitable interface thereof can rise to the other challenges put forward is worth considering.

6.5. Discussion

Its perhaps worth reconsidering the design principles behind SP outlined in Section 1 and relate it to the efforts above. As discussed, there is notable overlap, and by generalizing AMC to continuous domains, SP offers perhaps the simplest yet comprehensive formal picture for instantiating the large variety of computational problems considered in AI.

- on SP being *universal* and *generic*, approaches such as SMT and OMT are closely related, as they address problems from logical reasoning to some classes of mathematical programs, albeit the reasoning is restricted to fragments of first-order logic with arithmetic background theories. Classically, the solver technology as well as the specification syntax focused, not surprisingly, on finding satisfying assignments, and so the notion of counting as well as non-standard semantics (e.g., stable model semantics) is not handled. Relational models are also not typically considered. So, in some sense, SP = SMT + AMC over possibly non-standard semantics. While some constraint modeling languages might offer a better interface language – in the case of SP, only a basic interface was considered for illustration purposes – a unifying semantics to capture search, counting and optimization is not considered. Our view is that the semantics of programs is a paramount point when considering programming languages, so that issues such as composition and abstraction can be formally studied and properties proven [49].

- On SP being *declarative*, we borrowed the SMT syntax and introduced notions of semiring counting to capture the gamut of problems that can be expressed. This syntax is only offering a simple literal interpretation of the problem formalization as a SP program, but this is clearly not the only way to implement the semantics. So, ultimately realizing a functional interface [48, 77] or a relational interface [58], or even a combination may make the underlying framework more attractive.

  Discussions in [63] make it clear that while SP formalizes the fundamental computational tasks such as search, counting and optimization, we may need to consider additional machinery to compose complex learning paradigms, such as deep learning architectures, as well as data processing pipelines.

- On SP being *solver-independent* and *model-theoretic*, it closely follows the formulation of WMC, AMC and WMI. In that sense, SP = WMI + AMC over possibly non-standard and/or first-order domains (possibly with quantifiers).

  Of course, precisely because our understanding of compilation targets for continuous domains is not yet as mature as for discrete ones, an immediate solver strategy for WMI + AMC does not emerge, as it had for AMC. Moreover, because of the need for approximation in infinite domains, if such an extension were considered, it might not prove effective. Nonetheless, regardless of whether this encourages a single approximate scheme or a hybrid scheme, the semantics and formulation of SP is independent of the strategy.

When considering AMC in particular, we note that while it enjoys solver-independence and a model-theoretic formulation, it is not generic in the sense of handling SMT or WMI, for example, and not universal in the sense of handling optimization in continuous or non-factorized settings. Sum-product networks [44] are also not generic in the same manner, although continuous atoms are allowed as long as they can be decomposed and factorized, and treated essentially like propositions. In some sense, they are also not solver-independent, as they argue for algorithms implemented over a particular computational graph, but this is not necessarily a limitation as that is the very reason to use them. Overall, its perhaps best to view both AMC and sum-product networks as able to implement a (strict) fragment of SP.

7. Conclusions

In a nutshell, SP is a framework to declaratively specify four major concerns in AI applications:

- logical reasoning;
- non-standard models;
- discrete and continuous probabilistic inference;
- discrete and continuous optimization.

To its strengths, we find that SP is *universal* (in the above sense) and *generic* (in terms of allowing instantiations to particular logical languages and semirings). Thus, we believe SP represents a simple, uniform, modular and transparent approach to the model building process of complex AI applications.

SP comes with a *rigorous semantics* to give meaning to its programs. In fact, it is a semantic account that illustrates many search problems are instances of one underlying computational task. In that sense, we imagine future developments of SP would follow constraint programming languages and probabilistic programming languages in providing more intricate modeling features which, in the end, resort to the proposed semantics in the paper.

The most significant aspect of SP is that it also allows us to go beyond existing paradigms as the richness of the framework admits novel formulations that combine theories from these different fields, as illustrated by means of a combined regression and probabilistic inference example. Overall, this semantics not only encourages us to consider: (a) solution strategies that work for multiple domains and furthermore, combinations of such domains, (b) compositions of problem instances, and (c) new languages (e.g., the semantics of SP could as well be used for, say, non-standard models in continuous domains). In the long term, we hope SP will contribute to the bridge between learning and reasoning.

[1] M. Abo Khamis, H. Q. Ngo, and A. Rudra. Faq: questions asked frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 13–28, 2016.

[2] N. Andrei. *Nonlinear Optimization Applications Using the GAMS Technology*. Springer, 2013.

[3] U. Apsel, K. Kersting, and M. Mladenov. Lifting relational map-lps using cluster signatures. In *AAAI*, pages 2403–2409, 2014.

[4] R. A. Aziz, G. Chu, C. Muise, and P. J. Stuckey. Stable model counting and its application in probabilistic logic programming. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[5] F. Bacchus, S. Dalmao, and T. Pitassi. Solving #SAT and Bayesian inference with backtracking search. *J. Artif. Intell. Res.*, 34:391–442, 2009.

[6] C. Baral, M. Gelfond, and J. N. Rushton. Probabilistic reasoning with answer sets. *TPLP*, 9(1):57–144, 2009.

[7] J. S. Baras and G. Theodorakopoulos. Path problems in networks. *Synthesis Lectures on Communication Networks*, 3(1):1–77, 2010.

[8] C. Barrett, P. Fontaine, and C. Tinelli. The SMT-LIB Standard: Version 2.5. Technical report, Department of Computer Science, The University of Iowa, 2015. Available at `www.SMT-LIB.org`.

[9] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*, chapter 26, pages 825–885. IOS Press, 2009.

[10] V. Belle. Open-universe weighted model counting. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[11] V. Belle. Symbolic logic meets machine learning: A brief survey in infinite domains. *arXiv:2006.08480*, 2020.

[12] V. Belle and H. J. Levesque. Reasoning about discrete and continuous noisy sensors and effectors in dynamical systems. *Artificial Intelligence*, 262:189 – 221, 2018.

[13] V. Belle, A. Passerini, and G. Van den Broeck. Probabilistic inference in hybrid domains by weighted model integration. In *IJCAI*, 2015.

[14] V. Belle, A. Passerini, and G. Van den Broeck. Towards component caching in hybrid domains. In *AAAI*, 2016.

[15] D. L. Berre, E. Lonca, and P. Marquis. On the complexity of optimization problems based on compiled NNF representations. *CoRR*, abs/1410.6690, 2014.

[16] S. Bistarelli. *Semirings for Soft Constraint Solving and Programming*. SpringerVerlag, 2004.

[17] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint logic programming: syntax and semantics. *TOPLAS*, 23(1):1–29, 2001.

[18] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier. Semiring-based CSPs and valued CSPs: Frameworks, properties, and comparison. *Constraints*, 4(3):199–240, 1999.

[19] E. Boerger, E. Grädel, and Y. Gurevich. *The classical decision problem*. Springer Verlag, 1997.

[20] G. Brewka, J. P. Delgrande, J. Romero, and T. Schaub. asprin: Customizing answer set preferences without a headache. In *AAAI*, pages 1467–1474, 2015.

[21] G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[22] M. Cadoli and T. Mancini. Combining relational algebra, sql, constraint modelling, and local search. *Theory and Practice of Logic Programming*, 7:37–65, 1 2007.

[23] M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. *Artif. Intell.*, 172(6-7):772–799, 2008.

[24] D. Chistikov, R. Dimitrova, and R. Majumdar. Approximate counting in smt and value estimation for probabilistic programs. In *TACAS*, volume 9035, pages 320–334. 2015.

[25] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[26] L. De Raedt, A. Kimmig, and H. Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *Proc. IJCAI*, pages 2462–2467, 2007.

[27] R. Dechter and I. Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM (JACM)*, 50(2):107–153, 2003.

[28] G. V. den Broeck, N. Taghipour, W. Meert, J. Davis, and L. D. Raedt. Lifted probabilistic inference by first-order knowledge compilation. In *Proc. IJCAI*, pages 2178–2185, 2011.

[29] V. Derkinderen and L. De Raedt. Algebraic circuits for decision theoretic inference and learning. *ECAI*, 2020.

[30] C. H. Q. Ding, T. Li, and M. I. Jordan. Convex and semi-nonnegative matrix factorizations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(1):45–55, Jan. 2010.

[31] M. E. Dyer and A. M. Frieze. On the complexity of computing the volume of a polyhedron. *SIAM J. Comput.*, 17(5):967–974, 1988.

[32] J. Eisner and N. W. Filardo. Dyna: Extending Datalog for modern AI. In *Datalog Reloaded*, LNCS, pages 181–220. Springer, 2011.

[33] H. Enderton. *A mathematical introduction to logic*. Academic press New York, 1972.

[34] A. Ensan and E. Ternovska. Modular systems with preferences. In *IJCAI*, pages 2940–2947, 2015.

[35] R. Evans and E. Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

[36] H. Fargier, P. Marquis, and A. Niveau. Towards a knowledge compilation map for heterogeneous representation languages. In *IJCAI*, pages 877–883, 2013.

[37] H. Fargier, P. Marquis, and N. Schmidt. Semiring labelled decision diagrams, revisited: Canonicity and spatial efficiency issues. In *IJCAI*, 2013.

[38] D. Fierens, G. Van den Broeck, I. Thon, B. Gutmann, and L. De Raedt. Inference in probabilistic logic programs using weighted CNF's. In *UAI*, pages 211–220, 2011.

[39] D. Fontaine, L. Michel, and P. Van Hentenryck. Model combinators for hybrid optimization. In *CP*, pages 299–314, 2013.

[40] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL : a modeling language for mathematical programming*. Scientific Press, South San Francisco, 1993.

[41] E. Freuder. In pursuit of the holy grail. *Constraints*, 2(1):57–61, 1997.

[42] E. C. Freuder and A. K. Mackworth. Constraint satisfaction: An emerging paradigm. *Handbook of Constraint Programming*, pages 13–28, 2006.

[43] A. Friesen and P. Domingos. Recursive decomposition for nonconvex optimization. In *IJCAI*, 2015.

[44] A. Friesen and P. Domingos. The sum-product theorem: A foundation for learning tractable models. In *International Conference on Machine Learning*, pages 1909–1918, 2016.

[45] A. M. Frisch, W. Harvey, C. Jefferson, B. M. Hernández, and I. Miguel. Essence : A constraint language for specifying combinatorial problems. *Constraints*, 13(3):268–306, 2008.

[46] C. P. Gomes, A. Sabharwal, and B. Selman. Model counting. In *Handbook of Satisfiability*. IOS Press, 2009.

[47] J. Goodman. Semiring parsing. *Computational Linguistics*, 25(4):573–605, 1999.

[48] N. D. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: a language for generative models. In *Proc. UAI*, pages 220–229, 2008.

[49] A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani. Probabilistic programming. In *Proc. International Conference on Software Engineering*, 2014.

[50] M. Grant and S. Boyd. Graph implementations for nonsmooth convex programs. In *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008.

[51] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 31–40. ACM, 2007.

[52] P. Halmos. Measure theory. *Van Nostrad Reinhold Company*, 1950.

[53] J. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46(3):311–350, 1990.
[54] H. Hindi. A tutorial on convex optimization. In *American Control Conference*, volume 4, June 2004.
[55] J. N. Hooker and M. A. O. Lama. Mixed logical-linear programming. *Discrete Applied Mathematics*, 96-97:395–442, 1999.
[56] R. Kannan and C. L. Monma. *On the computational complexity of integer programming problems*. Springer, 1978.
[57] H. Kautz. Toward a universal inference engine. In *LPNMR*, volume 2923 of *LNCS*, pages 2–2. Springer Berlin Heidelberg, 2004.
[58] A. Kimmig, G. Van den Broeck, and L. De Raedt. An algebraic prolog for reasoning about possible worlds. In *Proc. AAAI*, 2011.
[59] A. Kimmig, G. Van den Broeck, and L. De Raedt. Algebraic model counting. *Journal of Applied Logic*, 22:46–62, 2017.
[60] S. Kolb, P. Z. D. Martires, and L. D. Raedt. How to exploit structure while solving weighted model integration problems. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, page 262, 2019.
[61] S. Kolb, M. Mladenov, S. Sanner, V. Belle, and K. Kersting. Efficient symbolic integration for probabilistic inference. In *IJCAI*, pages 5031–5037, 2018.
[62] D. Koller and N. Friedman. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press, 2009.
[63] P. Kordjamshidi, D. Roth, and K. Kersting. Systems ai: A declarative learning based programming perspective. In *IJCAI*, pages 5464–5471, 2018.
[64] R. A. Kowalski. Predicate logic as programming language. In *IFIP Congress*, pages 569–574, 1974.
[65] W. Kuich. Semirings and formal power series: their relevance to formal languages and automata. In *Handbook of formal languages, Vol. 1*, pages 609–677. Springer, Berlin, 1997.
[66] S. Lacoste-Julien, F. Huszár, and Z. Ghahramani. Approximate inference for the loss-calibrated bayesian. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 416–424, 2011.
[67] Z. Li and J. Eisner. First-and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 40–51. Association for Computational Linguistics, 2009.
[68] Y. Lierler and M. Truszczynski. An abstract view on modularity in knowledge representation. In *AAAI*, pages 1532–1538, 2015.
[69] G. Liu, T. Janhunen, and I. Niemelä. Answer set programming via mixed integer programming. In *KR*, 2012.
[70] G. Liu, T. Janhunen, and I. Niemelä. Introducing real variables and integer objective functions to answer set programming. In M. Hanus and R. Rocha, editors, *Declarative Programming and Knowledge Management*, volume 8439 of *Lecture Notes in Computer Science*, pages 118–135. Springer International Publishing, 2014.
[71] K. Marriott, N. Nethercote, R. Rafeh, P. Stuckey, M. Garcia de la Banda, and M. Wallace. The design of the zinc modelling language. *Constraints*, 13(3):229–267, 2008.
[72] B. Milch, B. Marthi, S. J. Russell, D. Sontag, D. L. Ong, and A. Kolobov. BLOG: Probabilistic models with unknown objects. In *Proc. IJCAI*, pages 1352–1359, 2005.
[73] T. Minka. Divergence measures and message passing. Technical report, Microsoft Research, 2005.
[74] D. G. Mitchell and E. Ternovska. A framework for representing and solving NP search problems. In *AAAI*, pages 430–435, 2005.
[75] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994.
[76] R. Nieuwenhuis and A. Oliveras. On sat modulo theories and optimization problems. In *Theory and Applications of Satisfiability Testing-SAT 2006*, pages 156–169. Springer, 2006.
[77] F. Obermeyer, E. Bingham, M. Jankowiak, D. Phan, and J. P. Chen. Functional tensors for probabilistic programming, 2019.
[78] F. Orsini, P. Frasconi, and L. De Raedt. kproblog: an algebraic prolog for machine learning. *Machine Learning*, 106(12):1933–1969, Dec 2017.
[79] H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE, 2011.
[80] M. Richardson and P. Domingos. Markov logic networks. *Machine learning*, 62(1):107–136, 2006.
[81] S. Sankaranarayanan, A. Chakarov, and S. Gulwani. Static analysis for probabilistic programs: inferring whole program properties from finitely many paths. *ACM SIGPLAN Notices*, 48(6), 2013.
[82] S. Sanner and D. A. McAllester. Affine algebraic decision diagrams (aadds) and their application to structured probabilistic inference. In *IJCAI*, pages 1384–1390, 2005.
[83] R. Sebastiani and S. Tomasi. Optimization modulo theories with linear rational costs. *ACM Trans. Comput. Logic*, 16(2):12:1–12:43, Feb. 2015.
[84] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. J. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *ICRA*, pages 639–646, 2014.
[85] E. Ternovska. An algebra of modular systems: Static and dynamic perspectives. In *International Symposium on Frontiers of Combining Systems*, pages 94–111. Springer, 2019.
[86] E. Ternovska and D. G. Mitchell. Declarative programming of search problems with built-in arithmetic. In *Proc. IJCAI*, pages 942–947, 2009.
[87] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.
[88] P. Van Hentenryck. Constraint and integer programming in OPL. *INFORMS Journal on Computing*, 14(4):345–372, 2002.
[89] P. M. Zuidberg Dos Martires, A. Dries, and L. De Raedt. Exact and approximate weighted model integration withprobability density functions using knowledge compilation. In *Proceedings of the 30th Conference on Artificial Intelligence*. AAAI Press, 2019.